ศรันย์ ไมตรีเวช

# วิชวลเบสิก กับวิศวกรรมซอฟต์แวร์

***วิชวลเบสิกในมุมมองที่จะนำมาใช้ประโยชน์สำหรับเป็นการพัฒนาซอฟต์แวร์ให้ใช้งานได้เป็นอย่างดี***

ปัญหาของการพัฒนาโปรแกรมหรือแอพพลิเคชันที่ยังแก้ไม่ตกตั้งแต่อดีตจนถึงปัจจุบันมีอยู่หลายอย่างหลายประการ นับเริ่มมาตั้งแต่การพยายามทำความเข้าใจกับความต้องการของผู้ใช้ให้ถูกต้องถ่องแท้ การนำความต้องการของผู้ใช้มาตีเป็นโจทย์เพื่อรวบรวมข้อมูลสู่ข้อสรุปว่า "เรา" ในฐานะผู้พัฒนาซอฟต์แวร์ จะต้องลงมือออกแบบและกำหนดขั้นตอนในการพัฒนาแอพพลิเคชันอย่างไรต่อไป

ประเด็นสำคัญอยู่ที่เมื่อเข้าสู่ขั้นตอนของการพัฒนา พูดง่าย ๆ ว่าเมื่อถึงเวลาโปรแกรมเมอร์ลงมือเขียนโค้ด หากแผนการต่าง ๆ ที่วางไว้ไม่ลงตัว มีจุดบอด ไม่ตรงกับความต้องการของผู้ใช้อย่างแท้จริง ปัญหาที่เกิดขึ้นจะน่าท้อใจเป็นอย่างยิ่ง เพราะการแก้โค้ดนั้นไม่ง่ายเหมือนแก้ผังออกแบบบนกระดาษ โดยเฉพาะอย่างยิ่งในสมัยที่ภาษาโปรแกรมยังมีความเกี่ยวเนื่องกันตั้งแต่หัวจดท้าย การแก้ไขอะไรบรรทัดเดียวอาจหมายถึงผลกระทบต่อระบบโปรแกรมเกือบครึ่งหรือทั้งหมด

เมื่อภาษาโปรแกรมถูกพัฒนาขึ้นมาเรื่อย ๆ ตามวันเวลาที่ผู้ผลิตภาษาโปรแกรมเล็งเห็นความสำคัญของการแบ่งโค้ดทั้งหมดออกเป็นส่วน ๆ แต่ละส่วนมีความเป็นเอกเทศที่สุดเท่าที่จะเป็นไปได้เพื่อความง่ายในการแก้ไขภายหลัง โปรแกรมเมอร์ก็เริ่มสบายขึ้น เขียนโค้ดได้เป็นระบบระเบียบขึ้น นำไปสู่ผลสรุปสุดท้ายคือ โปรแกรมที่ถูกปรับปรุงแก้ไขได้ง่าย โดยเฉพาะอย่างยิ่งหากผู้เขียนโปรแกรมได้จัดระเบียบของโค้ดไว้เป็นอย่างดี มีการเรียกใช้โพรซีเดอร์อิสระ (หรือที่เรียกไลบรารี) มาก ๆ โปรแกรมดังกล่าวจะมีลักษณะเป็นแผนที่ในตัวเอง แม้ผู้ที่มาแก้ไขภายหลังจะเป็นคนละคนกับผู้เขียน ก็จะรู้สึกเสมือนมีแผนที่ที่กางอ่านได้ง่ายบนฝ่ามือ จะเดินทางไปไหน เพื่ออะไรก็สะดวกง่ายดายไปหมด

มาเดี๋ยวนี้การเขียนโปรแกรมสมัยปัจจุบันยิ่งง่ายขึ้นไปอีก เมื่อโพรซีเดอร์หรือส่วนย่อยต่าง ๆ ในโค้ดถูกนำมารวมเข้ากับ "รูป" อินเตอร์เฟสที่ปรากฏจริงบนหน้าจอผู้ใช้ เมื่อเขียนโปรแกรมเสร็จแล้วและต้องการเข้าไปปรับปรุงแก้ไข โปรแกรมเมอร์เพียงดับเบิลคลิ๊กที่ตัวรูปอินเตอร์เฟสหนึ่ง ๆ แล้วจัดการปรับปรุงตัวโค้ดได้ทันที เช่น เมื่อผู้ใช้บ่นว่าคลิ๊กปุ่มจบการทำงานแล้วปรากฏว่าโปรแกรมปิดตัวเองทันที ทั้งที่บางครั้งการคลิ๊กดังกล่าวเกิดขึ้นจากความเผอเรอ เขาต้องการให้มีการดักถามเสียก่อนว่าต้องการให้จบการทำงานจริงหรือไม่ อย่างนี้โปรแกรมเมอร์เพียงดับเบิลคลิ๊กรูปปุ่มจบการทำงานเพื่อเข้าสู่หน้าจอแก้ไขโค้ดของปุ่มนั้น แล้วให้มีบรรทัดคำสั่งสร้างไดอะล็อกบ็อกซ์ถามย้ำความแน่ใจของผู้ใช้เสียก่อนเท่านี้ก็เป็นอันเสร็จสิ้นภารกิจ จะเห็นว่าเป็นงานแก้ไขที่แสนเบา ไม่ต้องมานั่งหาเหมือนแต่ก่อนว่าโพรซีเดอร์เพื่อการปิดโปรแกรมนั้นชื่ออะไร ตั้งอยู่ที่ไหน เมื่อหาเจอแล้วจะสร้างไดอะล็อกบ็อกซ์ที่เหมาะสมขึ้นมาได้อย่างไร

ในบรรดาเครื่องมือพัฒนาซอฟต์แวร์บนวินโดวส์ซึ่งเน้นการเขียนโค้ดที่ผูกติดกับออปเจ็กต์ หรือการมองโค้ดให้แยกเป็นสัดเป็นส่วนตามรูปวัตถุที่ผู้ใช้มองเห็นนั้น วิชวลเบสิกจัดได้ว่าเป็นเครื่องมือชิ้นแรกที่เป็นมิตร

อย่างยิ่งกับโปรแกรมเมอร์ จนถึงปัจจุบันไมโครซอฟต์ออกตัววิชวลเบสิกเวอร์ชัน 4.0 อย่างเป็นทางการแล้ว โดยส่วนตัวผมเห็นว่ามันน่าทึ่งที่สุดเท่าที่เคยเกิดการเปลี่ยนแปลงกับวิชวลเบสิกมาแต่ละเวอร์ชัน เพราะครั้งนี้ไม่เพียงเพิ่มคุณสมบัติและการใช้งานอันแสนวิเศษหลาย ๆ อย่างเข้ามา แต่ยังรวมไปถึงการออกแบบสภาพแวดล้อมที่ลดความขัดแย้งลง กลุ่มคำสั่งถูกรวมเป็นหมวดหมู่อย่างชัดเจนตามเป้าหมาย ผู้ที่เริ่มเข้ามาทำความรู้จักกับวิชวลเบสิกจะใช้เวลาน้อยกว่าเดิมในการทำความเข้าใจองค์ประกอบรวมทั้งหมด

แม้กล่าวได้ว่าวิชวลเบสิกเป็นเครื่องมือพัฒนาซอฟต์แวร์บนวินโดวส์ที่ใช้ง่าย กระนั้นมันก็ใช่จะง่ายอย่างที่เขาว่าปอกกล้วยเข้าปาก เพราะแม้จะไม่มีอะไรยุ่งยากซับซ้อนในวิถีทางการพัฒนาแอพพลิเคชัน ทว่ามันก็มีรายละเอียดให้จดจำเยอะแยะไปหมด ความจริงอันเป็นสัจจะอย่างหนึ่งก็คือ อะไรที่มีรายละเอียดให้จดจำมาก สิ่งนั้นยากจะเข้าถึงอย่างทะลุปรุโปร่งสำหรับบุคคลทั่วไป เนื่องจากข้อจำกัดในการเรียนรู้ ความฉลาดในการประยุกต์ และเวลาในการศึกษาเคสให้ทั่วถึงของแต่ละคนนั้นมีอยู่น้อย เรียนรู้ไปได้ครึ่งเดียวก็มีเวอร์ชันใหม่ที่ดีกว่า ใสปิ๊งกว่า ปลอดบักยิ่งกว่า และที่สำคัญมีรายละเอียดให้เรียนรู้เยอะกว่าออกมาสู่ตลาดเสียแล้ว

มีการพูดกันบ้าง ทำนองที่ว่าทฤษฎีเกี่ยวกับวิศวกรรมซอฟต์แวร์ (Software Engineering) ที่มีอยู่ในหลักสูตรสมัยโบราณ (โบราณในวงการคอมพิวเตอร์นี่อาจหมายถึงเมื่อห้าปีที่แล้วนี้เองนะครับ) นั้น ยังสามารถนำมาประยุกต์ใช้ได้กับเครื่องมือพัฒนาโปรแกรมหรือแอพพลิเคชัน อย่างเช่นวิชวลเบสิกในยุคนี้ได้หรือไม่?

จุดนี้เป็นเรื่องน่าสนใจมาก และเป็นประเด็นที่เราสามารถชี้ให้เห็นว่าความเป็นวิชวลเบสิก หรือเครื่องมือพัฒนาแอพพลิเคชันบนวินโดวส์อื่น ๆ ที่ขึ้นต้นด้วยคำว่า "วิชวล" นั้น มีความหมายในตัวเองอย่างไร

ก่อนอื่นมากล่าวถึงนิยามของวิศวกรรมซอฟต์แวร์เสียหน่อย มันคือระเบียบในการรวมวิธีการ เครื่องมือ และขั้นตอนมาประยุกต์เข้าด้วยกันเพื่อพัฒนาซอฟต์แวร์ หรือพูดง่าย ๆ ว่าวิศวกรรมซอฟต์แวร์ก็คือความฉลาดรัดกุมในกระบวนการผลิตซอฟต์แวร์นั่นเอง

โดยนิยามนี้ คุณจะไม่พบคำใดเกี่ยวข้องกับความก้าวหน้าหรือล้าหลังของเครื่องมือพัฒนาโปรแกรม เครื่องมือก็คือเครื่องมือ ความฉลาดในการใช้เครื่องมือต่างหากคือนิยามของคำว่าวิศวกรรม

การใช้สมองของมนุษย์มักเกี่ยวข้องกับการแยกระบบสักระบบออกเป็นส่วน ๆ คุณจะใช้คำว่า "วิเคราะห์" ก็ได้ เมื่อใดเราสามารถมองเห็นว่ารถมันวิ่ง พร้อมทั้งเข้าใจการทำงานประสานกันของกลไกทุกส่วนในเครื่องยนต์ เมื่อนั้นเราได้ชื่อว่าเข้าถึงวิศวกรรมยานยนต์ทำนองเดียวกัน เมื่อใดคุณสามารถเห็นว่าโปรแกรมมันรัน พร้อมทั้งเข้าใจว่ารากฐานการรันเริ่มมาจากความต้องการของผู้ใช้ ไล่มาถึงความสามารถในการจัดหาของผู้ผลิตซอฟต์-แวร์ ข้อจำกัดทางเครื่องมือพัฒนา ข้อจำกัดของฮาร์ดแวร์ ตลอดจนกระทั่งกลุ่มบุคลากรที่เข้ามาจัดการเรื่องราวในระดับโค้ด โค้ดของแต่ละคนให้ผลกับการรันโปรแกรมอย่างไร เมื่อนั้นคุณได้ชื่อว่าเข้าถึงวิศวกรรมซอฟต์แวร์

ฉะนั้นทฤษฎีวิศวกรรมซอฟต์แวร์จึงไม่เกี่ยวกับพ.ศ. แต่เกี่ยวกับความรู้ความเข้าใจว่าเครื่องมือที่อยู่ในมือเรานั้น ทำให้เรายืนอยู่ตรงจุดไหนของวิวัฒนาการ

ผมขอกล่าวถึงวิชวลเบสิกในมุมมองที่มันเป็นประโยชน์ต่อการพัฒนาซอฟต์แวร์ โดยแยกเป็นหัวข้อดังต่อไปนี้

## การสร้างแบบจำลองได้รวดเร็ว

ประเด็นนี้สำคัญอันดับหนึ่งในแง่ของปฐมบทแห่งการพัฒนาซอฟต์แวร์ นั่นคือ การออกแบบหน้าจอ สมัยก่อนนักวิเคราะห์ระบบกับผู้ใช้มักสื่อสารอย่างไม่อาจเข้าถึงกันเต็มร้อยเปอร์เซ็นต์ เพราะขาดเครื่องมือสำคัญที่จะช่วยให้เห็นว่า สิ่งที่ผู้ใช้ต้องการนั้น เมื่อถูกวาดลงบนหน้ากระดาษนักวิเคราะห์ กับเมื่อถูกพัฒนาขึ้นบนหน้าจอของโปรแกรมเมอร์แล้ว เป็นคนละเรื่องหรือใช่เรื่องเดียวกัน

สมัยก่อนการสร้างแบบจำลองขึ้นมาแต่ละหน้าจอนั้น เต็มไปด้วยอุปสรรค ใช้เวลาพอกับการเขียนขึ้นจริง หากผู้ใช้เห็นแบบจำลองแล้วเกิดความไม่ชอบใจ อยากขอให้เปลี่ยนใหม่ โปรแกรมเมอร์จะต้องใช้เวลาและความอดทนอย่างยิ่งยวดกว่าจะสร้างแบบจำลองที่ตรงตามความพอใจของผู้ใช้ขึ้นมาได้ แต่เมื่อถึงยุคของวิชวลเบสิกหรือเครื่องมือพัฒนาจำพวกวิชวลเหมือน ๆ กัน ขอเพียงนักวิเคราะห์มีความรู้ขั้นพื้นฐานการใช้งานมันเพียงพอ ก็อาจสร้างหน้าจอตามความต้องการของผู้ใช้ขึ้นมาต่อหน้าต่อตาเขาในเวลาเพียงห้านาทีหรือน้อยกว่านั้น เนื่องจากอินเตอร์เฟสส่วนใหญ่เป็นของสำเร็จรูปที่วิชวลเบสิกตระเตรียมไว้ให้ ผู้พัฒนามีหน้าที่เพียงดับเบิลคลิ๊กสร้างอินเตอร์เฟสจากทูลบ็อกซ์ขึ้นมาบนฟอร์ม จัดวาง ลากขยายหรือลดขนาดให้ได้สัดส่วนที่เหมาะสม เพียงเท่านี้ก็เป็นอันเสร็จสรรพ เราสามารถรันโปรแกรมให้ผู้ใช้ลองเล่น ลองคลิ๊ก ลองกระทั่งป้อนข้อมูลเข้าเท็กซ์บ็อกซ์ เพื่อหาความรู้สึกของตนเองว่าพอใจแล้วหรือยัง ถ้ายังเราก็ปรับปรุงแก้ไขหน้าจอไปเรื่อย ๆ ด้วยวิธีง่าย ๆ ปรับเปลี่ยนขนาดหรือหน้าตาออปเจ็กต์คอนโทรลด้วยเม้าส์หรือคีย์บอร์ด ผ่านตัวออปเจ็กต์คอนโทรลเอง หรือหน้าต่างพรอพเพอร์ตี้ส์

# การใช้คอนเส็ปต์การเขียนโค้ดเชิงวัตถุ

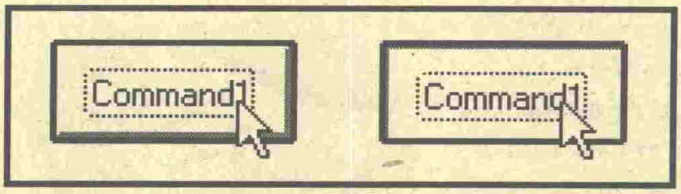
คอนเส็ปต์ในการเขียนโค้ดเชิงวัตถุ หรือที่เรียกว่า OOP (Object-Oriented Programming) นั้นนับได้ว่าเป็นก้าวกระโดดที่ยิ่งใหญ่ที่สุดในประวัติศาสตร์การทำซอฟต์แวร์ เนื่องจากเรานำโลกความเป็นจริงของพวกเราเข้าไปผสมกับการเขียนโค้ดมากขึ้น

ในโลกความเป็นจริงของพวกเรา เราอยู่กับสิ่งแวดล้อมที่เป็นวัตถุ วัตถุแต่ละอย่างมีสี ขนาด มวล และมีพฤติกรรมที่เป็นไปได้หลาก ๆ กัน อย่างเช่น เมื่อนึกถึงจักรยาน เรานึกถึงสิ่งประดิษฐ์มีโครงสร้างรูปสามเหลี่ยม มียางสองเส้นซึ่งเป็นตัวกำหนดความเป็นไปได้ที่จักรยานจะแสดงพฤติกรรม "เลื่อนตัว" ไปข้างหน้าข้างหลังได้อย่างสะดวก

จะเห็นว่าโลกความเป็นจริงในลักษณะนี้ เราเห็นได้ นึกถึงมันได้อย่างชัดเจน ทุกอย่างเป็นปกติยิ่งในระนาบสามัญสำนึกทั่วไป นักออกแบบเครื่องมือพัฒนาซอฟต์แวร์อย่างเช่นไมโครซอฟต์ก็พยายามนำเอาโลกของวัตถุแบบนี้ไปใส่ไว้ในคอมพิวเตอร์ กล่าวคือพยายามสร้างรูปอินเตอร์เฟสเพื่อการติดต่อกับผู้ใช้ให้หลอกตาได้เหมือนวัตถุจริง ๆ ยกตัวอย่างที่เห็นอยู่ทุกเมื่อเชื่อวันคือ ปุ่ม หรือที่เรารู้จักกันในนามของ Command Button ในวิชวลเบสิก ไมโครซอฟต์พยายามจำลองความเหมือนจริงแบบสามมิติให้เป็นสองมิติโดยใช้วิธีง่าย ๆ คือ ใช้สีขาว สีเทาสว่าง และสีเทาเข้มเข้ามาช่วย เมื่อมีการนำเม้าส์พอยน์เตอร์จากผู้ใช้มาคลิ๊ก ก็สลับสับเปลี่ยนความหนาของสีให้มองเห็นเสมือนปุ่มนั้นจมหายลงไป ดังรูปที่ 1

รูปที่ 1 นี้ก็ไม่มีอะไรมาก แค่ปุ่ม ๆ หนึ่งที่เมื่อนำเม้าส์พอยน์เตอร์ไปคลิ๊กแล้วปุ่มลงให้เห็น แต่เมื่อมองตามมุมมองของโปรแกรมเมอร์ ภาพที่เห็นนี้มีรายละเอียดปลีกย่อยแบ่งได้หลายอย่าง เช่น

**- เหตุการณ์ที่เกิดขึ้นจากผู้ใช้:** ตามความรู้สึกของผู้ใช้ เขาจะเห็นว่ามีเหตุการณ์เพียงหนึ่งเดียว คือ ตอนคลิ๊กเม้าส์ แต่โปรแกรมเมอร์ในวิชวลเบสิกจะมองออกมาว่ามีเหตุการณ์เกิดขึ้นอย่างน้อย 2 เหตุการณ์ คือเริ่มตั้งแต่เมื่อผู้ใช้นำพอยน์เตอร์มาลอยอยู่เหนือรูปปุ่ม ก็จะเกิดเหตุการณ์ Command1_MouseMove() ทันที หากมีโค้ดอยู่ในโพรซีเดอร์ดังกล่าว ก็จะเกิดการเอ็กซีคิวต์โค้ดทันที อีกเหตุการณ์คือ Command1_Click() เกิดขึ้นเมื่อผู้ใช้ทำการคลิ๊กเม้าส์ลงบนรูปปุ่ม อันนี้นำไปสู่ข้อสรุปที่ว่าโลกของโปรแกรมเมอร์จะอยู่เหนือโลกของสามัญสำนึก ภาษาโปรแกรมเชิงวัตถุเป็นภาษาที่ให้ความสนใจกับรายละเอียดทั้งหมดที่เกิดขึ้นกับรูปวัตถุ เราควบคุมทุกสิ่งให้เกิดขึ้นที่ไหน เมื่อไหร่ อย่างไรก็ได้ทั้งเป็นที่ประจักษ์และไม่ประจักษ์ต่อสายตาผู้ใช้




**รูปที่ 1 การใช้วิธีเปลี่ยนสีและขนาดของเส้นสีมาหลอกตาให้ผู้ใช้เห็นเหมือนรูปปุ่ม เป็นวัตถุที่จมหายลงไปในร่องได้จริง ๆ เมื่อเกิดการคลิ๊ก**

**- การปรากฏของรูปปุ่ม:** ตามการมองของผู้ใช้ เขาไม่คิดอะไรมากไปกว่าได้เห็นปุ่ม ๆ หนึ่ง เห็นแล้วเกิดความรู้สึกว่าถ้านำเม้าส์พอยน์เตอร์ไปคลิ๊ก มันควรจะยุบตัวลง แล้วเกิดปรากฏการณ์อะไรสักอย่างตามที่เห็นเขียนไว้บนหน้าปุ่ม เช่น ถ้าเห็นว่า "จบการทำงาน" อยู่บนหน้าปุ่ม เมื่อคลิ๊กแล้วก็ "น่า" ที่จะเกิดการปิดโปรแกรมลง แต่มุมมองของโปรแกรมเมอร์จะมองเข้าไปในรายละเอียดการปรากฏเป็นฉาก ๆ เริ่มตั้งแต่ขนาด ตำแหน่ง สีพื้นผิว อักขระบนหน้าปุ่ม และสีของอักขระนั้น ๆ เหล่านี้จะมีค่าดีฟอลต์กำหนดไว้ล่วงหน้าในหน้าต่างพรอพเพอร์ตี้ส์ โปรแกรมเมอร์สามารถเปลี่ยนได้ทั้งขณะก่อนรันและหลังรัน การปรากฏของออปเจ็กต์ที่เป็นอินเตอร์เฟสเหล่านี้ขึ้นอยู่กับศิลปะของโปรแกรมเมอร์แต่ละคน (หรือฝ่ายศิลป์ของแต่ละบริษัท) การจัดวางตำแหน่ง การกำหนดสัดส่วน การให้สีเหล่านี้มีผลให้ผู้ใช้เห็นและเข้าใจการใช้งานแอพพลิเคชันแตกต่างกันไป

ออปเจ็กต์แต่ละตัวจะมีชื่อประจำตัวโดยเฉพาะ ไม่ซ้ำกับออปเจ็กต์อื่น โดยชื่อนั้นจะเป็นพรอพเพอร์ตี้ส์หนึ่งของมัน นั่นคือ Name อย่างเช่นปุ่มทุกปุ่มเมื่อแรกสร้างจะมีชื่อดีฟอลต์เป็น Command1, Command2..., Command3 ตามลำดับ เมื่อเราจะกระทำการใด ๆ กับปุ่ม ก็เพียงอ้างชื่อและพรอพเพอร์ตี้ส์ที่ต้องการให้เปลี่ยนแปลง เช่น

```
Command1.Caption = "จบการทำงาน"
```

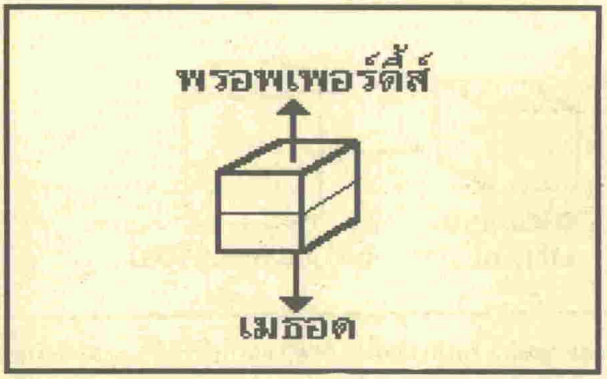
เมื่อเกิดการเอ็กซีคิวต์โค้ดบรรทัดนี้ เมื่อใด หน้าปุ่มซึ่งเคยเห็นว่าเป็น Command1 ก็จะเปลี่ยนไปเป็น "จบการทำงาน" ทันที

ทั้งผู้ใช้หรือแม้แต่โปรแกรมเมอร์เองอาจทำความเข้าใจกับคำว่า "วัตถุ" หรือออปเจ็กต์ในโปรแกรมด้วยการนึกถึงสิ่งที่ปรากฏให้เห็นง่าย ๆ ชัดเจนอย่างเช่นปุ่มที่ยกมาสาธิตนี้ แต่ในความเป็นจริงวัตถุเชิงโปรแกรมมิ่งไม่ได้จำกัดตัวเองอยู่แค่นั้น วัตถุในโลกของโปรแกรมอาจเป็นสิ่งที่ไม่ปรากฏต่อสายตาของผู้ใช้หรือโปรแกรมเมอร์ก็ได้

ความจริงการที่เรา "เห็น" ปุ่มนั้น เป็นแค่เพียงลักษณะที่ถูกกำหนดของวัตถุให้ "ถูกเห็นได้" ต่างหาก วัตถุในโลกของโปรแกรมไม่จำเป็นต้องมีรูปร่างอย่างนั้นอย่างนี้ สี่เหลี่ยมหรือวงกลมอะไร วิชวลเบสิกไม่ได้ให้ความสนใจเลย มันสนใจเพียงให้วัตถุมีคุณสมบัติหรือที่เรียกว่าพรอพเพอร์ตี้ส์ (Properties) มีพฤติกรรมเฉพาะตัว หรือที่เรียกว่าเมธอด (Methods) มีเหตุการณ์ที่เกิดขึ้นกับมันจากการกระทำของผู้ใช้และระบบ หรือที่เรียกว่าอีเวนต์ (Events)

ขอยกตัวอย่างที่ชัดเจนสักเรื่อง เรามาพูดถึงออปเจ็กต์ดาต้าเบสกัน ออปเจ็กต์ดังกล่าว

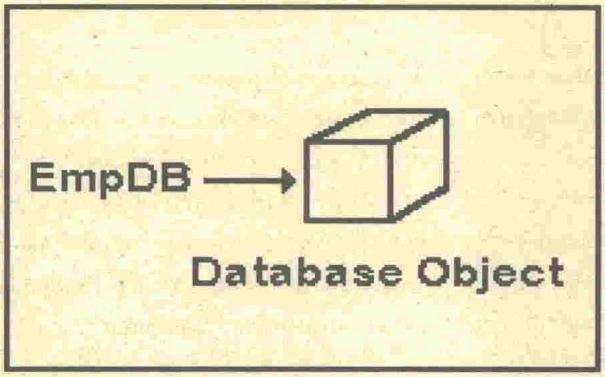
นี้ไม่มีรูปร่างปรากฏต่อสายตาของผู้ใช้หรือแม้แต่ตัวโปรแกรมเมอร์เอง เราได้แต่ "สมมติ" ให้มีออปเจ็กต์ดาต้าเบสขึ้นมาในโลกโปรแกรมมิง เอาเป็นว่าผมทึกทักให้ออปเจ็กต์ดาต้าเบสมีหน้าตาเป็นกล่องสี่เหลี่ยมลูกบาศก์ เพื่อให้เราสามารถสื่อสารถึงได้ระหว่างมโนนึกของมนุษย์ด้วยกัน เมื่อได้กล่องสี่เหลี่ยมลูกบาศก์ขึ้นมากล่องหนึ่งแล้ว ขั้นต่อไปเรามาดูว่ากล่องสี่เหลี่ยมนี้มันมีอะไรอยู่ข้างใน มาพูดกันในมิติของโปรแกรมมิง เราจะมองว่าวัตถุตัวนี้มีรายละเอียดอยู่สองชั้นซ้อนกัน คือคุณสมบัติผิวนอก (พรอพเพอร์ตี้ส์) และพฤติกรรมภายใน (เมธอด) ส่วนเหตุการณ์ที่เกิดขึ้นจากการกระทำของผู้ใช้หรือระบบนั้นไม่มี เนื่องจากไร้รูปร่อยใดให้ผู้ใช้เข้ามามีบทบาทโดยตรงผ่านเม้าส์หรือคีย์บอร์ด วาดได้ดังรูปที่ 2




รูปที่ 2
ออปเจ็กต์ดาต้าเบส
ในความนึกคิดคร่าว ๆ
ของมนุษย์

ขอย้ำว่าในความเป็นจริง ไม่ว่าจะโลกของเราหรือของโปรแกรมมิง ออปเจ็กต์ไม่ได้มีหน้าตาเป็นสี่เหลี่ยมหรือวงกลมอะไรทั้งนั้น ออปเจ็กต์หนึ่ง ๆ คือกลุ่มโค้ดที่ถูกเอนแคปซูเลต (Encapsulated) ไว้เป็นก้อนเอกเทศ ซึ่งเมื่อเข้ามาอยู่ในสภาพแวดล้อมที่เหมาะสมอย่างเช่นวิชวลเบสิก ออปเจ็กต์นั้นก็จะมีพรอพเพอร์ตี้ส์ เมธอด และเหตุการณ์ได้ตามที่มันถูกออกแบบกำหนดมาก่อน และไม่ว่าออปเจ็กต์จะถูกเห็นหรือไม่ก็ตาม สิ่งหนึ่งที่ขาดไม่ได้ในการทำให้มันปรากฏเป็นที่รับรู้ในหน้าจอเขียนโค้ดคือชื่อเฉพาะของมัน ชื่อเฉพาะดังกล่าวนี้ เมื่อประยุกต์กับออปเจ็กต์ประเภทที่เห็นได้ด้วยตาเช่น Command Button นั้น จะอยู่ในรูปของพรอพเพอร์ตี้ส์ Name ซึ่งคุณสามารถเห็นได้ในหน้าต่างพรอพเพอร์ตี้ส์เมื่ออยู่ที่โหมดออกแบบ แต่สำหรับชื่อเฉพาะของออปเจ็กต์ซึ่งมองไม่เห็นอย่างออปเจ็กต์ดาต้าเบสนี้ ทางเดียวที่เราจะอ้างถึงมันได้คือกำหนดตัวแปรขึ้นมาตัวหนึ่งอุปโลกน์ไว้ใช้แทนออปเจ็กต์ดาต้าเบส วิธีสร้างตัวแปรก็ไม่ได้ซับซ้อนพิสดารผิดแผกแตกต่างจากการประกาศตัวแปรทั่วไป เพียงคุณใช้คำสั่ง Dim และระบุชนิดตัวแปรให้เป็นดาต้าเบส เช่น

```
Dim EmpDB As Database
```




รูปที่ 3
เมื่อกำหนดตัวแปรให้เป็นออปเจ็กต์
ก็เหมือนเราผูกตัวแปรนั้น
กำหนดเงื่อนตายให้มันใช้แทน
ออปเจ็กต์หนึ่ง ๆ ได้เพียงอย่างเดียว

ในที่นี้ ตัวแปรชื่อ EmpDB ก็ถูกล็อกไว้เลยว่าต่อไปถ้าจะถูกนำมาใช้มีฐานะได้อย่างเดียวคือดาต้าเบสเท่านั้น กล่าวคือ EmpDB ไม่สามารถรับค่าตัวเลข ไม่สามารถรับค่าสตริง แต่จะมีพรอพเพอร์ตี้ส์ที่รับค่าเกี่ยวเนื่องกับดาต้าเบสได้ มีเมธอดที่ดำเนินการกับข้อมูลในดาต้าเบสได้

รูปที่ 3 เมื่อเราได้ตัวแปรซึ่งใช้แทนออปเจ็กต์ดาต้าเบสมาแล้ว เราก็ต้องทำให้ออปเจ็กต์ดังกล่าวเป็นตัวแทนของดาต้าเบสตัวใดตัวหนึ่ง จริง ๆ คุณจะเห็นว่าที่ผ่านมาเรายังไม่ดำเนินการอะไร ซึ่งจะเป็นการชี้ EmpDB ไปสู่ดาต้าเบสใดเลย วิธีการก็คือเราต้องใช้คำสั่ง Set

```
Set
EmpDB=OpenDatabase("Employee.MDB")
```

อย่างนี้ความเป็นออปเจ็กต์ดาต้าเบสของ EmpDB จะชี้ไปจับไฟล์ดาต้าเบสชื่อ Employee ซึ่งเป็นชนิดแอ็กเซสทันที ดูได้จากรูปที่ 4

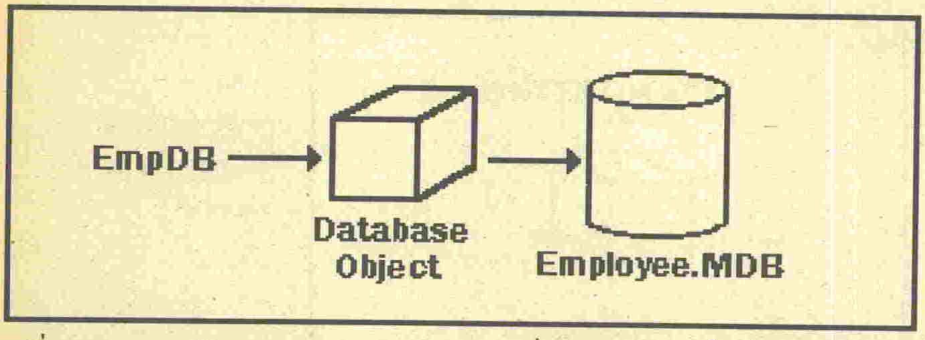
หลังจากที่เราได้ "ตัวแทน" ของ Employee.MDB ขึ้นมาแล้ว ต่อไปเราจะดำเนินการกับข้อมูลที่อยู่ภายในดาต้าเบส Employee .MDB อย่างไรก็ใช้ตัวแทนนี้เองนำไปสู่กระบวนการที่แท้จริง อย่างเช่นเราไม่ต้องการเทเบิลชื่อ Memo อีกต่อไป คือต้องการลบทิ้งจากดาต้าเบส ก็เพียงใช้คำสั่ง

```
EmpDB.TableDefs("Memo").Delete
```

จะเห็นว่า EmpDB ซึ่งเราเคยอุปโลกน์ให้มันเป็นตัวแปรชนิดดาต้าเบส บัดนี้มันมีพรอพเพอร์ตี้ส์ได้ เข้าถึงดาต้าเบสจริงได้ แล้วยังดำเนินการกับดาต้าเบสได้จริง ๆ อีกด้วย

จากตัวอย่างของออปเจ็กต์ดาต้าเบสที่ยกมาสาธิตนี้ คุณคงเข้าใจความหมายของ "วัตถุ" ในเชิงโปรแกรมมิงอย่างกระจ่างชัด พอสรุปได้ 3 ข้อใหญ่คือ

- ออปเจ็กต์อาจเป็นสิ่งที่คุณเห็นหรือไม่เห็นก็ได้
- เมื่อคุณจะอ้างถึงออปเจ็กต์ คุณต้อง




**รูปที่ 4 EmpDB จะเป็นตัวแทนของ Employee.MDB ต่อเมื่อเราเซตตัวแปรให้ชี้ไปที่ Employee.MDB ผ่านฟังก์ชัน OpenDatabase()**

มีชื่อให้อ้าง ไม่ว่าชื่อนั้นจะอยู่ในรูปของพรอพเพอร์ตี้ส์ Name หรือว่าตัวแปรออปเจ็กต์ซึ่งคุณตั้งขึ้นเอง

- ออปเจ็กต์ทุกตัวจะมีพรอพเพอร์ตี้ส์เมธอด หรือเหตุการณ์ อย่างใดอย่างหนึ่งหรือทั้งสามอย่าง เราสามารถเปลี่ยนแปลงรูปร่างหน้าตา ควบคุมพฤติกรรมของออปเจ็กต์ หรือใช้เหตุการณ์ใดเหตุการณ์หนึ่งซึ่งเกิดขึ้นกับออปเจ็กต์นั้น ๆ มาควบคุมหรือก่อให้เกิดกระบวนการอื่น ๆ ได้ตามใจชอบ

โดยนัยของความเป็นภาษาเชิงวัตถุของวิชวลเบสิกนี้ อาจแจกแจงประโยชน์ได้อย่างชัดเจนหลายหัวข้อ เช่น

1. โปรแกรมเมอร์ไม่จำเป็นต้องเขียนโค้ดด้วยระเบียบและขั้นตอนตายตัว ความนึกคิดในการเขียนโค้ดจะผูกติดอยู่กับออปเจ็กต์ที่อยู่บนฟอร์ม นับตั้งแต่ลักษณะการปรากฏภายนอก ไปจนถึงเหตุการณ์ที่เกิดขึ้นกับตัวออปเจ็กต์ เช่นนี้การค่อย ๆ เพิ่มออปเจ็กต์และโค้ดที่เกี่ยวเนื่องกับออปเจ็กต์นั้น ๆ เข้าไปตามแต่จะนึกได้ ก็อาจไม่ใช่เรื่องน่าตำหนิติเตียนหรือถูกรังเกียจว่าเป็นต้นเหตุทำให้เกิดความพัวพันยุ่งเหยิงในโค้ด แบบที่เรียกกันว่าสปาร์เก็ตตี้โค้ด แน่นอนว่าการเขียนโค้ดอย่างเป็นระเบียบรัดกุม มีการวางแผนล่วงหน้าเป็นอย่างดีย่อมน่าสรรเสริญกว่าการจับโค้ดใส่เพิ่มเข้าไปทีละเส้นสองเส้นตามอำเภอใจ แต่ประเด็นก็คือ การออกแบบนั้นถึงเลิศเลอเพียงใดก็ไม่อาจสมบูรณ์ที่สุดบนหน้ากระดาษ ดังนั้นเมื่อคำนึงถึงความจริงที่ว่าเราอาจต้องใส่เพิ่มหรือตัดออกได้อย่างไม่คาดฝันตลอดเวลา คุณลักษณ์อันให้ความยืดหยุ่นนี้ของวิชวลเบสิกจึงอาจทำให้คอนเส็ปต์ในการออกแบบไม่จำกัดตัวเองอยู่บนหน้ากระดาษเท่านั้น ทว่ายังยืดตัวมาได้ไกลถึงของจริง เพราะการลบทิ้งหรือเพิ่มออปเจ็กต์บางตัวเข้ามาบนฟอร์ม บางทีอาจง่ายเสียกว่าการลบเพิ่มบนหน้ากระดาษเสียอีก

2. ออปเจ็กต์มีพรอพเพอร์ตี้ส์อยู่หลายตัวที่มีค่าเป็นตัวเลขหรืออักษรที่ใช้อ้างอิงแทนตัวแปรได้ มันมีบทบาทสำคัญในการลดตัวแปรลง หากเราถือเอาหลักเกณฑ์ที่ว่าในโปรแกรมหนึ่งนั้น "ตัวแปรยิ่งน้อยเท่าไหร่ยิ่งดี" อันนี้จะเห็นประโยชน์ของพรอพเพอร์ตี้ส์ได้ชัดเจน

ประโยชน์ของการสร้างตัวแปรให้น้อยที่สุดได้แก่

- ลดความขัดแย้งในการเขียนโปรแกรมเอง รวมทั้งจำนวนบรรทัดที่ใช้เขียนโค้ดด้วย
- ใช้ทรัพยากรน้อยลง
- ปรับปรุงโปรแกรมในภายหลังได้ง่ายขึ้น (เพราะมีสิ่งที่จะนำมาพิจารณาทบทวนน้อยลง มีสิ่งที่ต้องตัด แทรก เสริมน้อยลง)

ในความเป็นจริงผมพบว่าโปรแกรมเมอร์ส่วนใหญ่เลือกวิธีสะดวกคือ การกำหนดตัวแปรขึ้นมาใช้เฉพาะหน้า โดยไม่ไตร่ตรองหาหนทางที่ดีกว่านั้น ตัวอย่างที่ง่ายที่สุดได้แก่ การเปลี่ยน Row ของ Grid เช่น เมื่อคลิ๊กปุ่ม ChangeGridRow ก็จะเกิดการป้อนข้อมูลทีละเรคอร์ด

```
Sub ChangeGridRow_Click ()
  Static n As Integer
  If n = Grid1.Rows-1 Then Exit Sub
   n = n + 1
  Grid1.Row = n
End Sub
```

ถ้าเขียนอย่างนี้จะเห็นได้ว่าเราต้องประกาศตัวแปร n ขึ้นมาเป็นการเปลืองเนื้อที่หน่วยความจำ และเป็นการสิ้นเปลืองอย่างถาวรตลอดการรันด้วย เนื่องจากต้องทำให้เป็น Static ซึ่งจะรักษาค่าไว้แม้ออกจากโพรซีเดอร์ไปแล้ว อีกประการคือ บรรทัดของโค้ดต้องเพิ่มขึ้นสองบรรทัด แทนที่จะเขียนได้สั้น ๆ อย่างนี้

```
Sub ChangeGridRow_Click ()
      If Grid1.Row = Grid1.Rows-1
      Then Exit Sub
      Grid1.Row = Grid1.Row + 1
End Sub
```

แม้ตัวอย่างจะดูง่ายและเห็น ๆ กันอย่างนี้แหละครับ ขยันสร้างตัวแปรขึ้นมาใช้กันนักแล แทนที่จะอ้างอิงจากค่าพรอพเพอร์ตี้ส์เดิม ในการใช้งานจริงซึ่งคุณต้องใช้ออปเจ็กต์คอนโทรลหลาย ๆ ตัว มันจะมีค่าพรอพเพอร์ตี้ส์ที่เกี่ยวกับพอยน์เตอร์ทำนองนี้อยู่เยอะขึ้น ถ้าหากคุณไม่ครุ่นคิดค้นหากันจริง ๆ บางทีอาจทำให้ต้องเสียพื้นที่หน่วยความจำให้กับตัวแปรที่ไม่จำเป็นอะไรเลยเอาง่าย ๆ แม้จะเป็นโปรแกรมเมอร์มือฉมังก็ตาม บางทีเกิดอาการ "คิดไม่ถึง" ได้เหมือนกัน วิธีที่ดีคือ หมั่นนึกบ่อย ๆ เกี่ยวกับเรื่องนี้

เกี่ยวกับการกำจัดตัวแปรฟุ่มเฟือยชนิดตัวเลขแล้ว มาพูดถึงตัวแปรฟุ่มเฟือยชนิดอักขระกันบ้าง หลายคนไม่ทราบครับว่ามีพรอพเพอร์ตี้ส์พิเศษอยู่ตัวหนึ่งในออปเจ็กต์ทุกตัว ไม่เว้นแม้แต่ฟอร์ม ที่สามารถทำหน้าที่บันทึกข้อมูลประจำตัวออปเจ็กต์นั้น ๆ เป็นอักขระได้ พรอพเพอร์ตี้ส์ที่ว่านี้คือ Tag ยกตัวอย่างที่เห็นได้ง่ายและต้องใช้กันเป็นประจำคือการเก็บข้อมูลอินพุตเก่าในเท็กซ์บ็อกซ์ ปกติเราจะประกาศตัวแปรขึ้นมาใช้เก็บเช่น

```
Option Explicit
Dim LastFocus As String
Dim LastInputOfText1,
   LastInputOfText2,
   LastInputOfText3 As String
```

→

```
Sub Text1_KeyPress (KeyAscii As Integer)
      If KeyAscii = 13 And Text1 <> "" Then
            LastFocus = "Text1"
            InputOfText1 = Text1.Text
      End If
End Sub

Sub TakeBack ()
      Select Case LastFocus
            Case "Text1"
                  Text1 = LastInputOfText1
            Case "Text2"
                  Text2 = LastInputOfText2
            Case "Text3"
                  Text3 = LastInputOfText3
      End Select
End Sub
```

ด้วยโค้ดข้างต้นนี้ เมื่อผู้ใช้เขาคีย์อินพุตเข้าไปที่ Text Box ตัวใดตัวหนึ่งแล้วเคาะเอนเตอร์ จะทำให้เกิดการเก็บข้อมูลปัจจุบันไว้ (ความจริงอาจมีการใช้ข้อมูลอินพุตตัวนี้ทำอะไรอย่างอื่นอีกเยอะ) เมื่อเขาคีย์ข้อมูลชุดใหม่ แต่ยังไม่เคาะเอนเตอร์ เขามีสิทธิ์ที่จะเรียกข้อมูลเดิมก่อนเปลี่ยนแปลงขึ้นมาได้ด้วยโพรซีเดอร์ TakeBack ซึ่งอาจเป็นโค้ดใน Command Button หรือเป็นสับอิสระอย่างไรก็แล้วแต่เราจะออกแบบกำหนด จะเห็นว่ายิ่งคุณมี Text Box มาก คุณก็ยิ่งต้องประกาศตัวแปรไว้ใช้เยอะขึ้นเป็นเงาตามตัว หากโปรแกรมของคุณจำเป็นต้องใช้ตัวแปรเยอะอยู่แล้ว ตัวแปรฟุ่มเฟือยเหล่านี้ก็ยิ่งเข้ามาบั่นทอนทรัพยากรร่อยหรอลงไปอีก เวลาแก้โค้ดก็นั่งปวดหัวไล่เรียงนานขึ้นกว่าเดิม แทนที่จะเป็นอย่างนั้น เรามาใช้พรอพเพอร์ตี้ส์ Tag เข้าช่วยจะดีกว่าครับ โค้ดทั้งหมดจะเขียนใหม่ได้ดังนี้

```
Option Explicit
Dim LastFocus As String

Sub Text1_KeyPress (KeyAscii
As Integer)
      If KeyAscii = 13 And Text1
      <> "" Then
            LastFocus = "Text1"
            Text1.Tag = Text1.Text
      End If
End Sub

Sub TakeBack ()
      Select Case LastFocus
        Case "Text1"
            Text1 = Text1.Tag
        Case "Text2"
            Text2 = Text2.Tag
        Case "Text3"
            Text3 = Text3.Tag
      End Select
End Sub
```

3. โดยที่ออปเจ็กต์ในโลกของโปรแกรมดำรงอยู่ด้วยลักษณะสามัญเป็นชนิด ๆ ดังนั้นจึงเป็นไปได้ที่เราจะสร้างออปเจ็กต์ใหม่เลียนออปเจ็กต์แม่แบบโดยถือเอาชนิดหรือ "เผ่าพันธุ์" เดียวกันเป็นเกณฑ์ เช่น Command Button นั้นจะสร้างสักกี่ปุ่มมันก็ออปเจ็กต์เดียวกัน ต่างแค่คุณสมบัติภายนอกเท่านั้น เนื้อหาภายในไม่ต่างกันเลย เมื่อมีแม่แบบขึ้นมาตัวหนึ่งก็มีตัวลูกตามมาได้ วิชวลเบสิกเพียงคัดลอกทุกสิ่งมาจากต้นฉบับ ยกเว้นก็แต่หมายเลขอินเด็กซ์ ตัวแม่จะเป็น 0 ตัวลูกจะเป็น 1, 2, 3 ตามลำดับการเกิด คอนเส็ปต์ที่น่าทึ่งนี้ทำให้เราได้ศัพท์เฉพาะคือ "คอนโทรลอะเรย์" ขึ้นมา ทำนองเดียวกันกับที่เมื่อก่อนเราใช้ตัวแปรอะเรย์มาแล้วจนคุ้น

การใช้คอนโทรลอะเรย์มีความหมายอย่างยิ่งยวด มันเป็นประโยชน์ข้อสำคัญของงานโปรแกรมมิงเชิงวัตถุ เพราะ

- ลดจำนวนโค้ดซ้ำ ๆ กันให้เหลือน้อยที่สุด เช่น แทนที่คุณจะต้องเขียนโค้ดอย่างที่เห็นในโพรซีเดอร์ Text1_KeyPress() อีกใน Text2_KeyPress() และ Text3_KeyPress() ก็จะเหลือโค้ดเพียงกลุ่มเดียว เป็นการลดปริมาณโค้ดได้ถึงสามเท่าตัว (ถ้ายิ่งมี Text Box รับงานเดียวกันนี้มากขึ้น ปริมาณของโค้ดก็น้อยลงเป็นอัตราส่วนแปรผันตรง)

- นำไปใช้ร่วมกับตัวแปรอะเรย์ซึ่งสร้างไว้เป็นพอยน์เตอร์กี่ชุดก็ได้ไม่จำกัด เป็นประโยชน์ต่องานแสดงข้อมูลที่มีลำดับฟิลด์ในเทเบิลหลากหลาย โดยเฉพาะงานแสดงเอาต์พุตที่มาจากเทเบิลเดียวกันแต่ไม่เรียงตามลำดับฟิลด์ของเทเบิลนั้น

ตัวอย่างของประโยชน์ในข้อแรกเขียนเป็นโค้ดได้ดังนี้

```
Option Explicit
(none)

Sub Text1_KeyPress (Index As Integer,
KeyAscii As Integer)
      If KeyAscii = 13 And Text1
      (Index) <> "" Then
            Text1(Index).Tag =
            Text1(Index).Text
      End If
End Sub

Sub TakeBack ()
Dim x As Integer
  For x = 0 To 2
   Text1(x).Text = Text1(x).Tag
  Next x
End Sub
```

จะเห็นว่าด้วยการโค้ดแบบนี้ คุณไม่จำเป็นต้องเขียนโค้ดซ้ำ ๆ กันใน Text2, Text3, ..., TextN เพียงเขียนใน Text1 ซึ่งเป็นคอนโทรลอะเรย์โพรซีเดอร์เดียวจบ

4. เราสามารถอ่านโค้ดได้ง่ายขึ้น อันเนื่องจากอินเตอร์เฟสปรากฏเป็นตัว ๆ ในลักษณะของออปเจ็กต์คอนโทรล ซึ่งคอนโทรลแต่ละตัวก็มีกลุ่ม มีชนิดของตนเองเป็นที่แน่นอน เราก็สามารถถือความดีข้อนี้เป็นการตั้งชื่อให้เกิดความอ่านง่าย มองเข้าไปเห็นทั้งชนิดและหน้าที่ของคอนโทรลในคราวเดียวกันจากชื่อในโค้ดนั้นเอง

ตัวอย่างเช่น เรามีปุ่มอยู่มากมายในโปรแกรม หากแต่ละปุ่มมีชื่อหลาก ๆ กันไปตามใจชอบ เช่น Delete, ShowForm2, Cancel, Update เราก็คงจำไม่ได้ว่าชื่อของคอนโทรลที่เห็นนี้มีชนิดเป็นอะไร ยิ่งหากนำไปรวมกับคอนโทรลอื่นในโค้ดก็ไปกันใหญ่ หากเราเติมพรีฟิกซ์เข้าไปเพียงสามตัว ผลที่ได้จะแตกต่างกันอย่างมหาศาลกับการตั้งชื่อขึ้นมาลอย ๆ ตามอำเภอใจ เช่น cmdDelete, cmdShowForm2, cmdCancel, และ cmdUpdate อย่างนี้เราจะเกิดความหมายรู้ขึ้นมาทันทีว่าคอนโทรลทั้งสี่ตัวเป็นออปเจ็กต์ชนิดเดียวกันคือ Command Button แต่ละปุ่มมีภารกิจประการใดก็ประกาศชัดอยู่แล้วในชื่อสามัญของพวกมันเอง แถมยังมีผลพลอยได้พ่วงท้าย คือเมื่อลิสต์รายชื่อคอนโทรลดูในหน้าต่างเขียนโค้ด คอนโทรลแต่ละชนิดจะจับกลุ่มอย่างเป็นระเบียบ คุณสามารถเลือกหาได้เร็วขึ้น มีความเข้าอกเข้าใจความสัมพันธ์ของออปเจ็กต์ในโค้ดมากขึ้น

ในลักษณะดังกล่าวนี้ หากเป็นภาษาสมัยเก่า คุณไม่สามารถถือประโยชน์จากการใช้พรีฟิกซ์ได้มากนัก เนื่องจากตัวแปรแต่ละตัวไม่เคยชี้ไปที่ออปเจ็กต์ร่วมชนิดเลย

# หาข้อผิดพลาดของโค้ดแบบครบวงจร

การตรวจสอบความสมบูรณ์ของโปรแกรมตามทฤษฎีแล้ว จุดประสงค์คือการค้นให้พบข้อผิดพลาดของโปรแกรม พูดง่าย ๆ ว่าจับผิดมันเสียก่อนที่มันจะผิดให้ผู้ใช้พบ โดยกระบวนการแล้ว อาจสรุปสั้น ๆ ว่า เราต้องตรวจสอบอย่างพิถีพิถัน นับแต่ความสมบูรณ์ของโค้ดแต่ละบรรทัด ความสมบูรณ์ของเอาต์พุตที่จะออกมาตรงตามกำหนด ไปจนกระทั่งการทำงานประสานกับระบบใหญ่ที่โปรแกรมต้องเข้าไปเกี่ยวข้อง

ในขั้นตรวจสอบความสมบูรณ์ของโค้ด ไม่ว่าจะเป็นเรื่องของไวยากรณ์ การใช้ตัวแปรผิดพลาด ตลอดไปจนกระทั่งการอ้างชื่อโพรซีเดอร์ซ้ำซ้อน ปัจจุบันไม่ใช่หน้าที่ของมนุษย์อีกต่อไป เราไม่จำเป็นต้องคำนึงเรื่องเหล่านี้เลยหากเครื่องมือพัฒนามีความเป็น Compiler-Interpreter อย่างที่วิชวลเบสิกเป็นอยู่ กล่าวคือ เมื่อเขียนโค้ดหนึ่งบรรทัดเสร็จก็มีการตรวจสอบเดี๋ยวนั้นว่าโครงสร้างไวยากรณ์ผิดไหม มีการประกาศตัวแปรก่อนใช้หรือยังอะไรทำนองนี้ เราได้รู้ทันทีทันใดโดยไม่ต้องรันทั้งโปรแกรมหรือคอมพายล์ให้ได้ไฟล์ EXE เสียก่อน

ยกตัวอย่างเช่น เมื่อคุณพิมพ์ว่า If x = 12 แล้วเคาะเอนเตอร์ทันที หมายความว่าคุณเขียนผิดไวยากรณ์ของโครงสร้าง If เข้าแล้ว เพราะบรรทัดเดียวกับ If จะต้องมี Then อยู่เสมอ อย่างนี้ก็จะมีเมสเสจบ็อกซ์ขึ้นมาเตือนทันทีว่าคุณลืมใส่ Then ข้างหลังตัวเลข 12

ในขั้นตรวจสอบความสมบูรณ์ของเอาต์พุตว่าออกมาตรงกับออกแบบไว้หรือไม่ เราก็อยู่ในโลกของการ "เกิดเหตุการณ์ตามการกระทำจากผู้ใช้" หรือที่เรียกเป็นศัพท์เฉพาะว่า Event Driven ในโลกแบบนี้ช่วยเหลือเราได้มาก เพราะสิ่งที่ผู้ใช้จะกระทำก็คือ สิ่งที่เห็นอยู่ชัด ๆ บนฟอร์ม เราเช็คได้ทีละเหตุการณ์ ไม่ใช่ว่ามีการเอ็กซีคิวต์โค้ดไหลเรียงกันจากต้นจนจบเหมือนภาษาโปรแกรมสมัยก่อน เมื่อพบความผิดพลาดจากเหตุการณ์ใด ก็เข้าไปแก้ไขที่เหตุการณ์นั้น ๆ โดยไม่กระทบกระเทือนกับเหตุการณ์อื่น (เว้นแต่จะมีความสัมพันธ์กันระหว่างสองเหตุการณ์ ซึ่งนั่นก็ง่ายที่จะตรวจสอบ เนื่องจากกลไกการเรียกโพรซีเดอร์อื่นนั้นต้องระบุเป็นชื่ออย่างชัดเจน หากันได้อย่างรวดเร็ว)

นอกจากความเป็น Event Driven แล้ว วิชวลเบสิกยังมีอุปกรณ์ช่วยเหลือในการตรวจสอบค้นหาข้อผิดพลาด หรือที่เรียกกันว่า Debugging Tools ซึ่งส่วนใหญ่มุ่งหาค่าตัวแปรในจังหวะเหตุการณ์ที่เราเลือก อย่างเช่น เราต้องการรู้ว่าเมื่อผู้ใช้คลิ๊กปุ่ม OK ตัวแปร X มันมีค่าเป็นเท่าไหร่ ควรอยู่ในย่านที่เหมาะสมไหม เราก็โฟกัสลงไปเลยว่า X ที่โพรซีเดอร์ CommandOk_Click() มันมีค่าเป็นอะไร ขอให้แสดงให้ดูในโหมดพักการรันชั่วคราว (Break Mode) การทำให้อยู่ในโหมดพักการรันทำได้หลายวิธี ตั้งแต่การใช้คำสั่ง Stop ลงไปในโค้ด (เมื่อจัดเก็บไฟล์จะปนไปกับโค้ดด้วยจริง ๆ) และการใช้ปุ่ม F9 สร้าง Break Point ขึ้นมาชั่วคราว

ในทางปฏิบัติเราอาจจะต้องรันไปหยุดพักไป และตรวจสอบค่าตัวแปรทีละบรรทัดบ้าง ทีละโพรซีเดอร์บ้าง วิชวลเบสิกก็อนุญาตให้เรารันโปรแกรมได้หลายแบบ ทั้งแบบทีละบรรทัดและแบบทีละโพรซีเดอร์ (หมายถึงเมื่อมีการเรียกโพรซีเดอร์จากบรรทัดหนึ่ง ให้เอ็กซีคิวต์ทุกบรรทัดของโพรซีเดอร์นั้นแล้วจึงหยุดอีก) อย่างนี้ทำให้การตรวจสอบไม่เยิ่นเย้อ การสันนิษฐานจุดผิดพลาดถูกตรวจสอบได้รวดเร็ว

ถึงแม้จะมีผู้ช่วยตรวจสอบอยู่หลายตัว แต่สิ่งที่ยังขาดไม่ได้ก็คือ ตามนุษย์ธรรมดา ๆ นี่แหละครับ เป็นสิ่งเดียวในปัจจุบันที่ใช้พิจารณาว่าสิ่งที่ปรากฏบนหน้าจอตรงตามเป้าหมายที่วางไว้ หรือออกแบบไว้หรือไม่ อันนี้คงไม่มีซอฟต์แวร์หรือฮาร์ดแวร์ที่ไหนมาช่วยเราได้ (อย่างน้อยก็ในพ.ศ.นี้)

ส่วนในขั้นตรวจสอบการทำงานร่วมกับระบบใหญ่ อันนี้ผู้ผลิตเครื่องมือพัฒนาโปรแกรมมักจะมีการตรวจสอบการประสานงานผิดพลาดขั้นพื้นฐานไว้ให้ เมื่อดักพบข้อผิดพลาดหนึ่ง ๆ ก็จะมีการบอกกล่าวกันในรูปของเมสเสจบ็อกซ์ ระบุให้ทราบเป็นประเด็นไปว่าเกิดอะไรขึ้นกับการเชื่อมต่อระบบ ไม่ใช่รายงานคร่าว ๆ ว่า "การเชื่อมต่อล้มเหลว" หรือที่คล้ายอย่างนี้

## สรุป

ผมเองเคยพัฒนาแอพพลิเคชันบนดอสมาก่อน นับเริ่มตั้งแต่ยุคที่ต้องเขียนโค้ดเป็นภาษาเบสิกบนเครื่องแอปเปิ้ล แล้วค่อยขยับพัฒนามาเป็นซี ปาสคาล และโคบอลบนเครื่องไอบีเอ็มคอมแพตฯ ซาบซึ้งดีว่าภาษาต่าง ๆ มีอรรถรสอย่างไร ว่ากันว่าความรู้ความเข้าใจภาษาโปรแกรมแต่ละภาษานั้นเกิดขึ้นเมื่อตอนแก้บักหากคุณทำให้โปรแกรมของคุณปลอดบักสำเร็จเมื่อไหร่ เมื่อนั้นได้ชื่อว่าคุณเขยิบความรู้ความเข้าใจภาษาดังกล่าวเพิ่มขึ้นมาอีกนิด

เมื่อเริ่มยุควินโดวส์ฉบับไมโครซอฟต์ซึ่งทั่วโลกให้ความสนใจจากน้อยไปหามากและจากมากไปหามากที่สุดถึงปัจจุบัน เชื่อเถอะครับว่าแอพพลิเคชันที่ใช้งานทั่วไปในปัจจุบันหันมารันบนวินโดวส์กันหมดแล้ว เท่าที่หลงเหลืออยู่บนดอสเห็นจะได้แก่เกมและยูทิลิตี้ซึ่งไม่มีความจำเป็นต้องพึ่งพาสภาพแวดล้อมของวินโดวส์ แต่เมื่อจอมเผด็จการอย่างวินโดวส์ 95 ออกตัวมาแล้ว ต่อไปแนวโน้มที่ซอฟต์แวร์ "ทุกตัว" จะต้องรันให้ได้บนวินโดวส์ก็เริ่มเป็นความจริงขึ้นมาบ้างแล้ว เนื่องจากผู้ที่ลงวินโดวส์ 95 ไว้ในเครื่องจะไม่อย่างในดอสโหมดโดยดีฟอลต์แต่แรกเริ่มอีกต่อไป ผู้ที่รันเกมหรือยูทิลิตี้บนดอสจะเกิดความรู้สึกอึดอัดกว่าแต่ก่อนเยอะแม้มีวิธีเลี่ยงไปบูตด้วยไดรฟ์อื่นก็ตาม

ดังนั้นจึงไม่มีโปรแกรมเมอร์คนไหนจะขัดขืนฝืนต้านกระแสอันเชี่ยวกรากของวินโดวส์ได้ หากยังดื้อที่จะพัฒนาซอฟต์แวร์บนดอส ก็คาดหมายได้เลยว่าผลงานนั้นทำเองใช้เองมากกว่าที่จะจำหน่ายหรือติดตั้งให้ผู้ใช้ที่ไหน เนื่องจากธรรมชาติของผู้ใช้นั้น เมื่อคุ้นเคยกับการใช้งานรูปแบบใด ก็จะยึดติดอยู่กับรูปแบบนั้น ไม่ยอมโยกย้ายไปเป็นอื่น ปัจจุบันผู้ใช้นับแต่วัยเยาว์ขึ้นไปถึงรุ่นอาวุโสต่างก็ติดรูปแบบการใช้งานบนวินโดวส์กันอย่างถอนตัวไม่ขึ้น เมื่อพวกเขาพบว่าการคลิ๊กเม้าส์นั้นง่ายกว่าการพิมพ์คำสั่งด้วยคีย์บอร์ดตั้งหลายร้อยหลายพันเท่า

และเมื่อพูดในแง่โปรแกรมมิงเอง ก็ไม่เห็นจะมีเครื่องมือพัฒนาบนดอสตัวไหนที่ให้คุณค่าของโปรแกรมเชิงวัตถุได้ดีเท่าเครื่องมือบนวินโดวส์ โปรแกรมเมอร์ที่ยังติดอยู่กับอำนาจอันยิ่งใหญ่ของภาษาซีที่เข้าถึงทรัพยากรได้ทั่วถ้วน ก็ยังต้องเปลี่ยนใจเมื่อพบว่าฟังก์ชัน API ของวินโดวส์ก็มีอำนาจเดียวกัน หรืออาจจะเหนือกว่าไว้ให้ด้วย ไม่ขาดหายไปไหน

ผมอยากจะพูดว่าแนวโน้มในการพัฒนาซอฟต์แวร์ยุคต่อไปควรจะอยู่ในมือของวินโดวส์ หรือคอนเส็ปต์แบบเดียวกับวินโดวส์เว้นไว้แต่จะเป็นซอฟต์แวร์ประเภทที่อยากได้ความเร็วในการเข้าถึงทรัพยากรมาก ๆ โปรแกรมเมอร์บางรายอาจยังรักที่จะใช้ซีหรือแม้กระทั่งแอสเซมบลีอยู่ ความยากง่ายในการโค้ดก็แน่นอนว่าจะต้องเป็นคนละเรื่องกับการเขียนโค้ดบนวินโดวส์

ไม่แน่นะครับ ต่อไปคำว่า Dos อาจปรากฏเป็นคำศัพท์ในพจนานุกรมทำนองว่า "ครั้งหนึ่ง คอมพิวเตอร์เคยทำงานบนดอส..." อะไรอย่างนี้ก็เป็นได้ ไม่ช้าไม่นานนี้แหละ

**บริษัท ซิสเทกซ์ อินโฟโปร จำกัด**
**SYSTEX INFOPRO COMPANY LIMITED**
**ประกาศ**
**รับสมัครงาน**

บริษัท ซิสเทกซ์ อินโฟโปร จำกัด เป็นผู้นำในการจัดจำหน่ายอุปกรณ์ DISPLAY DEVICES, ระบบคอมพิวเตอร์ และให้บริการข้อมูลซื้อขายหลักทรัพย์ขณะทำการ (REAL-TIME) พร้อมซอฟต์แวร์ มีความต้องการรับสมัครงานหลายตำแหน่งตามคุณสมบัติ และจำนวนอัตราดังนี้

1. **พนักงานฝ่ายขาย** ตำแหน่ง Sales Representative
   - วุฒิ ปวส.-ปริญญาตรี (สาขาบริหารธุรกิจ, การตลาด)
   - อายุ 20 ปีขึ้นไป, มีความรู้ทางด้านภาษาอังกฤษพอสมควร
   - มีความรู้ทางด้านหลักทรัพย์, คอมพิวเตอร์จะได้รับการพิจารณาเป็นพิเศษ
2. **พนักงานบัญชี**
   - วุฒิ ปวส.-ปริญญาตรี, อายุ 20 ปีขึ้นไป
   - ประสบการณ์ทำงานอย่างน้อย 1 ปี
   - มีความรู้ทางด้านคอมพิวเตอร์จะได้รับการพิจารณาเป็นพิเศษ
3. **โปรแกรมเมอร์**
   - วุฒิปริญญาตรีทางคอมพิวเตอร์, อายุ 20 ปีขึ้นไป
   - มีประสบการณ์เขียนภาษา C หรือ PASCAL
   - มีประสบการณ์ด้าน COMMUNICATION จะได้รับการพิจารณาเป็นพิเศษ
4. **ช่างเทคนิค**
   - วุฒิ ปวช.-ปวส. สาขาอิเล็กทรอนิกส์ (รับจำนวนมาก)
   - มีความรู้ทางด้านคอมพิวเตอร์, พ้นภาระการเกณฑ์ทหารแล้ว
5. **เจ้าหน้าที่สต๊อกสินค้า**
   - วุฒิ ปวช ขึ้นไป (สาขาบัญชีหรือสาขาที่เกี่ยวข้อง) ใช้เครื่อง PC ได้
   - อายุ 20 ปีขึ้นไป

บริษัทฯ ให้ผลตอบแทนที่ดี มีค่าครองชีพ เบี้ยขยัน ประกันชีวิต ประกันสุขภาพ โบนัส และสวัสดิการอื่น ๆ ทุกตำแหน่งหากมีประสบการณ์จะได้รับการพิจารณาเป็นพิเศษ ถ้าเป็นชายต้องพ้นภาระการเกณฑ์ทหารแล้ว ผู้สนใจกรุณาส่งจดหมาย หรือมาสมัครด้วยตนเองได้ที่

**บริษัท ซิสเทกซ์ อินโฟโปร จำกัด**
1168/42-43 ชั้น 17 อาคารลุมพินีทาวเวอร์
ถนนพระราม 4 แขวงทุ่งมหาเมฆ เขตสาธร
กรุงเทพฯ 10120
โทร. 285-6660-1, 285-6282, 679-8140-8
**โบนัส 3-6 เดือน**